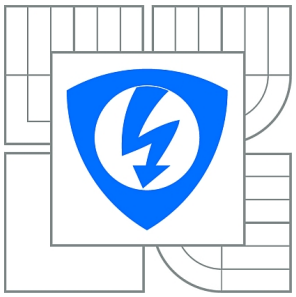
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

SEGMENTACE MR OBRAZŮ POMOCÍ ALGORITMŮ STROJOVÉHO UČENÍ

SEGMENTATION OF MR IMAGES USING MACHINE LEARNING ALGORITHMS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

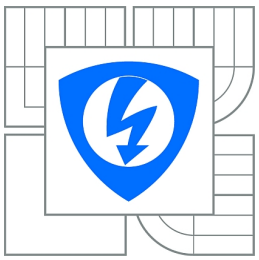
AUTOR PRÁCE — JAN DORAZIL

AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL DVOŘÁK

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Jan Dorazil	***ID:***	154696
Ročník:	3	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Segmentace MR obrazů pomocí algoritmů strojového učení

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte algoritmy strojového učení a jejich využití pro automatickou segmentaci MR obrazů. Otestujte různé příznaky a jejich vhodnost pro vybraný algoritmus a zadaná vstupní data. Vytvořte nástroj, který pomocí zadaných trénovacích dat provede segmentaci vstupního obrazu.

DOPORUČENÁ LITERATURA:

[1] Christopher M. Bishop: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] Hastie, T., Tibshirani, R., Friedman, J.: The elements of statistical learning: data mining, inference and prediction, Springer, 2 edition, (2009)

Termín zadání:	9.2.2015	***Termín odevzdání:***	2.6.2015

Vedoucí práce: Ing. Pavel Dvořák
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zabyvá segmentací snímků magnetické rezonance za použití algoritmu Random Forests. Využívané technologie při plnění práce zahrnují programovací jazyk C++ s knihovnami ITK a OpenCV. Práce popisuje postup zpracování obrazu od jeho načítání, předzpracování až po samotnou segmentaci. Výsledkem práce je program, který plně automaticky segmentuje MR snímky hlavy myši na mozek a okolí.

KLÍČOVÁ SLOVA

MR, segmentace, random forest, strojové učení, ITK, OpenCV

ABSTRACT

This thesis concerns with magnetic resonance image segmentation using Random Forests algorithm. Employed technologies accomplishing the specified task include C++ programming language with libraries ITK and OpenCV. This work descibes the technique of processing images from loading through preprocessing to the actual segmentation. The outcome from this work is a programme that automatically segmentates MR images of mouse's head to the brain and the surroundings.

KEYWORDS

MR, MRI, segmentation, random forest, machine learning, ITK, OpenCV

DORAZIL, Jan *Segmentace MR obrazů pomocí algoritmů strojového učení*: bakalářská práce. BRNO: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 50 s. Vedoucí práce byl Ing. Pavel Dvořák

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Segmentace MR obrazů pomocí algoritmů strojového učení“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

BRNO ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Pavlu Dvořákovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěl poděkovat své přítelkyni a své rodině za korekturu textu práce a hlavně morální podporu, bez které bych jistě tuto práci nemohl dokončit.

BRNO . . . . . .

. . . . . .

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Tato práce se zabývá segmentací MR (Magnetická Rezonance) snímků hlavy myši za použití ML (strojové učení – Machine Learning) algoritmu *Random Forests*. Výsledkem segmentace by měl být snímek segmentovaný na mozek myši a jeho okolí.

Segmentace obrazu je proces, při kterém se původně celistvý obraz dělí na jednotlivé části (segmenty), které je dále možné zpracovávat odděleně. Tento proces je jeden z nejdůležitějších kroků zpracování obrazů, obzvláště pak pokud se jedná o medicínské obrazy. Z pohledu člověka je segmentace obrazu triviální činnost, se kterou se setkáváme každý den — za segmentaci lze považovat i rozeznávání objektů, které vidíme. Automatická segmentace pomocí počítače však často triviální nebývá, a pro získání použitelných výsledků je třeba věnovat určitý čas volbě algoritmu pro danou situaci. Metody segmentace se dělí podle nutnosti zásahu člověka na:

- manuální segmentace
- poloautomatická segmentace
- automatická segmentace

Při *manuální segmentaci* je celý proces segmentace prováděn člověkem buďto za použití specializovaného počítačového softwaru, či přímo na snímku. *Manuální segmentace* bývá často velmi časově náročná, obzvláště pokud se jedná o vícerozměrné snímky. *Poloautomatická segmentace* je jakýsi mezistupeň mezi manuální a automatickou segmentací. Při poloautomatické segmentaci člověk přibližně označí jednotlivé segmenty snímku ručně a tyto data dále projdou automatizovaným procesem, jehož výstupem je segmentovaný obraz. Automatická segmentace bývá pro implementaci a často i výpočetní výkon počítače nejnáročnější. Při tomto typu segmentace jsou všechna získaná data zpracovávána automaticky za pomocí počítače.

Ačkoli obrazová data pro segmentaci mohou být získávána nepřeberným množstvím způsobů, tato práce se zabývá pouze zpracováním medicínských obrazů. Mezi nejčastější způsoby získávání medicínských obrazů patří především sonografie, rentgen, CT (počítačová tomografie – Computed Tomography) a MR (Magnetická Rezonance). Data, o kterých se pojednává v této práci, pocházejí z *magnetické rezonance*.

1 TEORETICKÁ ČÁST

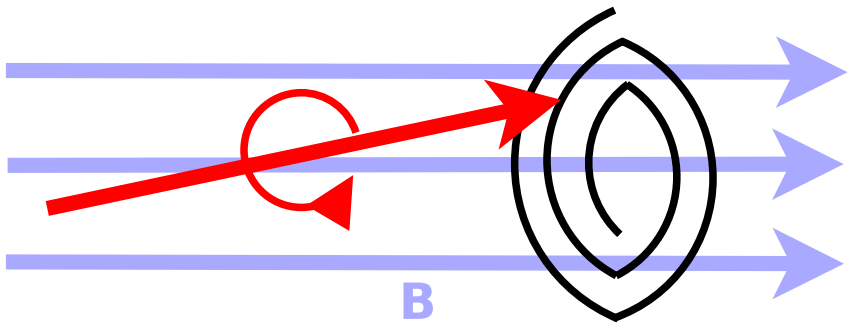
1.1 Princip magnetické rezonance

Znalost principu funkce MR není pro tuto práci příliš důležitá. Zpracovávaná data však pocházejí právě z magenetické rezonance, a proto zde má vysvětlení jejích principů své místo.

Magnetická rezonance využívá pro svou funkci faktu, že jádra atomů, ze kterých se skládá tkáň, mají samy o sobě určitý moment hybnosti (tzv. *spin*). Když se rotující jádro vystaví magnetickému poli, osa rotace jádra se postupně vyrovná se směrem působení magnetického pole. Předtím, než se vyrovná, nastane určitý přechodový děj. Osa rotace jádra se začne pohybovat po spirále, dokud se neustálí (viz. Obr. 1.1).

Pokud ovšem pozorujeme velké množství atomových jader, ze kterých se skládá skenovaná tkáň, zjistíme, že působení statického magnetického pole na jednotlivá jádra je minimální. Orientace os jader se bude zdát stále náhodná díky vzájemnému ovlivňování jednotlivých jader v tkáni. Přesto však bude možné pozorovat slabou tendenci jader k orientaci ve směru magnetické indukce zavedeného pole. Po určité době se pohyb jader ve tkáni ustálí a jádra se dostanou do stabilního stavu.

Zavedením dalšího, tzv. *transverzálního* pole lze některá jádra z tohoto stavu vyvést a osy jader začnou znovu rotovat. Nastavením správné velikosti statického a transverzálního pole je možné vybírat, o která jádra se bude jednat. Jádra, která jsou vyvedena ze stabilního stavu pak vydávají slabé magnetické pole, jež určitou dobu přetrvává, a to i po odpojení cívek transverzálního pole. Snímáním tohoto pole získáme obrazová data, která lze zpracováním na počítači převést na výsledný snímek.[6]



Obr. 1.1: Rotace jádra

Modrý vektor ukazuje směr statického magnetického pole, červený vektor je osa rotace jádra atomu.

1.2 Úvod do strojového učení

1.2.1 Základní pojmy

Tato část se zabývá základy strojového učení, které jsou nutným předpokladem k probírání algoritmů využitých v této práci.

Strojové učení je obor zabývající se způsoby umožnujícími počítačovému systému „učit se“. Podle způsobu učení se algoritmy strojového učení dělí na:

- učení s učitelem —*supervised learning*
- učení bez učitele —*unsupervised learning*

Algoritmy strojového učení s učitelem pracují se vstupními (trénovacími) daty, ke kterým jsou přiřazeny požadované výstupy. Každá dvojice vektoru vstupních proměnných a vektoru výstupních proměnných se nazývá *pozorování*. Zpracováním těchto dat pak algoritmus rozpoznává určité závislosti mezi vstupními a požadovanými výstupními daty. Díky těmto závislostem je algoritmus schopen na dalších vstupních datech předpovídat výstupy. Úspěšnost tohoto algoritmu se pak hodnotí podle odchylky předpovídaných výstupů a výstupů očekávaných.

Algoritmy strojového učení bez učitele nemají ke vstupním datům přiřazeny žádné výstupy. Tyto algoritmy tedy hledají v datech nějakou logickou strukturu. Lze je využít v situaci, kdy máme zadané velké množství dat, jevících se na první pohled náhodně, a hledáme v nich určité pravidlo, případně modelujeme, jak asi tato data vznikla.[4]

Podle výsledků, které mají tyto algoritmy poskytnout se dělí na:

- Regresní algoritmy
- Klasifikační algoritmy

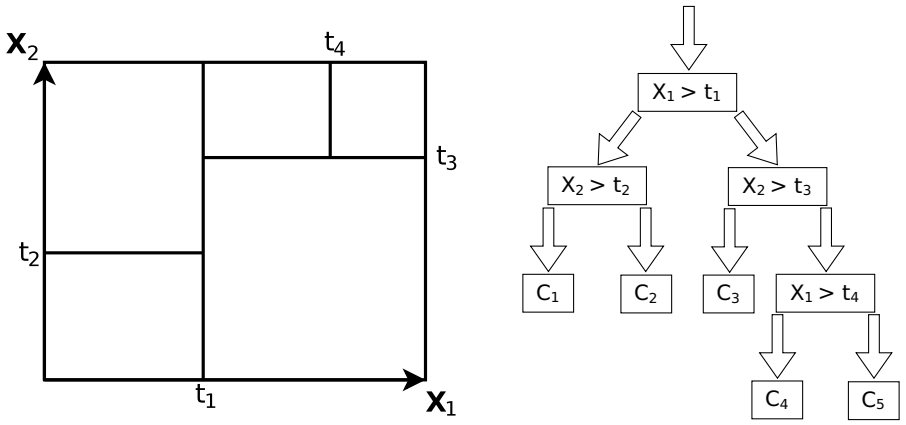
Výstupy regresních algoritmů jsou spojité a tyto algoritmy se tedy na základě vstupních hodnot snaží aproximovat funkci výstupních hodnot. Výstupy algoritmů klasifikačních jsou již diskrétní. Klasifikační algoritmy se snaží kombinaci vstupních hodnot přiřadit určitou *třídu*, kdy počet tříd bývá standardně předem daný. [4]

Podoborem strojového učení je tzv. „dolování dat“ (data mining), které se zabývá praktickou implementací statistických algoritmů strojového učení. Algoritmů pro dolování dat existuje nepřeberné množství, pro tuto práci jsou však nejdůležitější algoritmy založené na stromových datových strukturách.

1.2.2 Stromové algoritmy

Stromové algoritmy strojového učení mají svůj základ v datových strukturách binárních stromů. Binární strom je struktura, která má jeden „kořen“ a ten se rekurzivně

dělí na dvě „větve“ (viz. Obr. 1.2(b)). Pro lepší představu funkce stromových algoritmů je vhodné pracovat se dvěmi vstupními proměnnými a jednou výstupní proměnnou. Tyto dvě vstupní proměnné společně tvoří dvourozměrný prostor, ve kterém lze zobrazit jednotlivá pozorování, která jsou při trénování použita. Aplikací stromového algoritmu pak dělíme tento prostor rekurzivně na další a další podprostory, kde každé rozvětvení stromu (node) reprezentuje dělící přímku kolmou k jedné z os vstupních proměnných. Každému výslednému podprostoru je poté přidělena určitá funkce, která aproximuje v daném prostoru hodnotu výstupní proměnné. Tato funkce je specifikována zakončujícími položkami stromu (tzv. *leave node*). Nejjednodušší volbou aproximující funkce může být konstanta. Predikce výstupní proměnné pak probíha tak, že postupujeme v binárním stromu od kořene, a v každém rozvětvení pokračujeme doprava nebo doleva, pokud je podmínka v daném rozvětvení splněna, respektive nesplněna.[4] Obrázek 1.2(a) zobrazuje rekurzivní dělění prostoru způsobené aplikací binárního stromu z obrázku 1.2(b).



(a) Rekurzivně děleny prostor (b) Binární strom

Obr. 1.2: Prostor rekurzivně dělený binárním stromem

Při implementaci stromových algoritmů je důležitá volba proměnné, na které bude dělení provedeno, a také hodnota této proměnné, která bude mezní hodnotou oddělující dva podprostory. Dělíme-li prostor na M podprostorů pak $R_1, R_2, ..., R_M$ označují množiny, do kterých patří vstupní proměnné jednotlivých podprostorů. Indexy L a R značí dva podprostory R_L a R_R, které vznikly rozdělením jednoho prostoru na levý a pravý podprostor. Pro získání indexu dělící vstupní proměnné j a dělícího bodu s hledáme řešení následující minimalizace:

$$min_{j,s}[Q_L + Q_R], \tag{1.1}$$

kde

$$N_m = \#\{x_i \in R_m\}, \tag{1.2}$$

$$Q_m = \sum_{x_i \in R_m} (y_i - c_m)^2 \tag{1.3}$$

a

$$c_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i. \tag{1.4}$$

Rovnice 1.3 značí tzv. *node impurity measure* (x_i jsou vstupní a y_i výstupní proměnné). Uvedený impurity measure se používá pro regresní problémy. Pro klasifikační problémy lze použít například *Gini index* (viz. vzorec 1.5).[4]

$$Q_m = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{1.5}$$

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{1.6}$$

1.2.3 Bootstrap aggregation

Nevýhodou stromových algoritmů je jejich nestabilita způsobená vysokým rozptylem *predikovaných* (předpovídaných) hodnot. Za tento problém je zodpovědný samotný princip fungování algoritmu —nevhodná volba místa dělení prostoru, na počátku tvorby binárního stromu, vede k dalším chybám v nižších vrstvách stromu. K nevhodné volbě může dojít chybou ve výstupní proměnné jediného pozorování.[4]

Ke zvýšení stability stromových, ale také jiných, algoritmů se používá *bootstrap aggregation* nebo zkráceně *bagging*. Tato metoda je alterací metody *bootstrap*, která se využívá pro určení statistických údajů o daném modelu. Například určení rozptylu predikovaných hodnot modelu $S(\mathbf{Z})$ použitím trénovacích dat $\mathbf{Z} = (z_1, z_2, ..., z_N)$, kde z_i je dvojice x_i a y_i, lze provést pomocí metody bootstrap takto:

- Vytvoříme B vektorů $\mathbf{Z}^{*b}$ z trénovacích dat pomocí *náhodného vybírání s opakováním* (lze vybrat vícekrát jeden prvek) o velikosti N.
- Použitím B bootstrap vektorů jako trénovacích dat trénujeme daný model.
- Výsledný rozptyl spočteme podle následujícího vzorce:

$$Var[S(\mathbf{Z})] = \frac{1}{B-1} \sum_{b=1}^{B} (S(\mathbf{Z}^{*b}) - \overline{S}^*)^2, \tag{1.7}$$

kde

$$\overline{S}^* = \sum_b \frac{S(\mathbf{Z}^{*b})}{B}. \tag{1.8}$$

Použití metody bagging pro stromové algoritmy funguje na stejném principu. Nejprve vytvoříme z trénovacích dat B bootstrap vektorů o velikosti N pomocí náhodného vybírání s opakováním z původních trénovacích dat. Dále pak použitím B boostrap vektorů trénujeme B regresních či klasifikačních stromů, čímž získáme

modely $S(\mathbf{Z}^{*b})$. Predikci výstupní proměnné pro vstupní proměnnou x budeme provádět tak, že určíme nejprve predikci každého z B modelů standardním způsobem, čímž získáme predikce $f^*(x)$. Dále pak z těchto predikcí určíme výslednou predikci celé skupiny stromů. Pro regresní problémy určíme výslednou predikci jako průměrnou hodnotu všech predikcí $f^*(x)$, a pro klasifikační problémy vybereme tu predikci, která se mezi $f^*(x)$ vyskytuje nejčastěji. Výsledkem baggingu je snížení rozptylu výsledných predikcí. Pokud stanovíme, že výsledná predikce je náhodná veličina, která má rovnoměrné rozdělení a jednotlivé predikce z B stromů jsou nezávislé pak rozptyl výsledné predikce baggingu je:

$$\frac{1}{B}\sigma^2, \tag{1.9}$$

kde σ^2 je rozptyl predikce za použití jednoho stromu. Pokud však jednotlivé predikce nejsou nezávislé náhodné veličiny pak rozptyl výsledné predikce bude:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2, \tag{1.10}$$

kde ρ je korelační koeficient mezi jednotlivými predikcemi. Čím větší bude počet bagging stromů B, tím menší bude hodnota druhého členu. První člen však zůstává vždy stejný. Lze tedy usoudit, že vzájemná korelace mezi jednotlivými bagging stromy zamezuje redukci rozptylu, a tím také omezuje funkčnost baggingu.[4]

1.3 Random Forests

1.3.1 Historie

Random forests[1] je metoda určená pro regresní i klasifikační problémy, kterou společně vyvíjeli Leo Breiman a Adele Cutler. Metoda byla zpočátku ovlivněna výzkumy Yali Amit a Donalda Gemana, kteří přišli s nápadem hledat, při konstrukci binárního stromu, z náhodně vytvořené podmnožiny původních vstupních proměnných. Tento princip je v algoritmu Random Forests stěžejní.[5]

Popis tohoto algoritmu byl poprvé uveden v dokumentu od Lea Breimana. Tento dokument popisuje metodu vytváření skupiny stromů s využitím algoritmu podobnému CART společne s užitím metody baggingu. Na dalším vývoji tohoto algoritmu dnes pracuje Microsoft Research.[5]

1.3.2 Princip

Random forests je dalším z řady algorimů strojového učení. Jde o jistou modifikaci metody bagging. Random forests vytváří při trénování soustavu B binárních stromů,

[1] Leo Breiman, 2001

kde každý jednotlivý strom je trénován bootstrap vektorem, který vznikl z původních trénovacích dat. Tím se snižuje rozptyl predikce stromů, což vede ke zlepšení stability. Další změnou, která byla v Random forests algoritmu zavedena je v samotném trénování stromů. Před každým hledáním vhodného bodu, pro rozdělení prostoru vstupních proměnných, je z p vstupních proměnných náhodně vybráno m ($m < p$), se kterými se bude pokračovat. Výběrem m vstupních proměnných se snižuje korelace mezi jednotlivými bagging stromy, čímž se redukuje hlavní problém, který metoda bagging má. Proces Random forests lze tedy po krocích zobrazit takto:

1. Z vektoru trénovacích dat o velikosti N získáme bootstrap vektor $\mathbf{Z}^{*b}$ o velikosti N.
2. Rekurzivně opakujeme následující kroky na každé konečné větvi stromu, dokud nezískáme větev pro níž je velikost podprostoru menší nebo rovna n_{min}.
   (a) Vybereme náhodně z p vstupních proměnných m.
   (b) Vybereme nejlepší kombinaci j (index vstupní proměnné) a s (bod rozdělení) z m vybranných proměnných.
   (c) V daném bodě rozdělíme prostor vstupních proměnných na dva podprostory.
3. Výsledkem trénování je skupina B stromů.

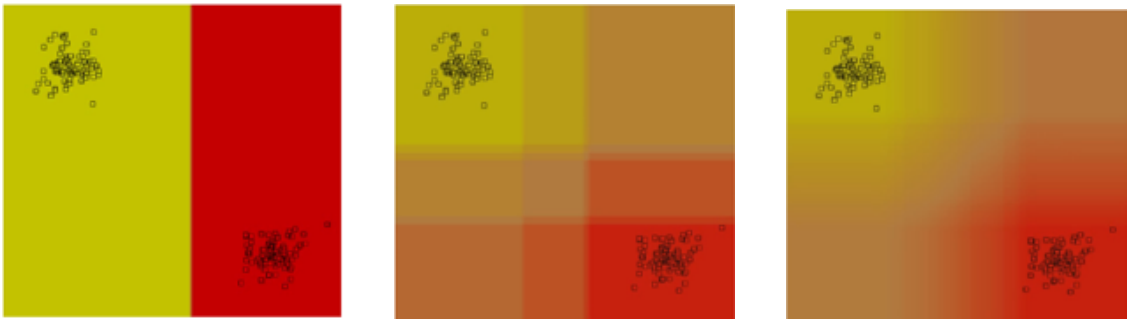
Predikce se ze skupiny stromů provádí stejně jako u baggingu: pro regresní problémy aritmetickým průměrem predikcí jednotlivých stromů, pro klasifikační problémy hlasováním jednotlivých stromů.[4]

1.3.3 Vliv parametrů algoritmu

Vliv počtu použitých stromů

Počet stromů B je parametrem algoritmu, který přímo ovlivňuje tvar predikční funkce. Každý jednotlivý strom aproximuje výstupní data soustavou konstantních funkcí. Výsledná predikční funkce jednotlivých stromů tedy nebude hladká. Výpočtem průměrné hodnoty predikce většího počtu stromů, kde každý strom má zavedenou určitou náhodilost do svých trénovacích dat však získáváme funkci hladkou. Čím více stromů tedy bude použito, tím hladší bude průběh výsledné funkce.[5]

Predikční funkce pro klasifikační problémy však nemůže být hladká. U klasifikačních problémů počet stromů ovlivňuje pouze to, jak hladká je funkce zobrazující pravděpodobnost klasifikace vektoru hodnot vstupních proměnných do určité třídy k (viz. Obr. 1.3).[5]

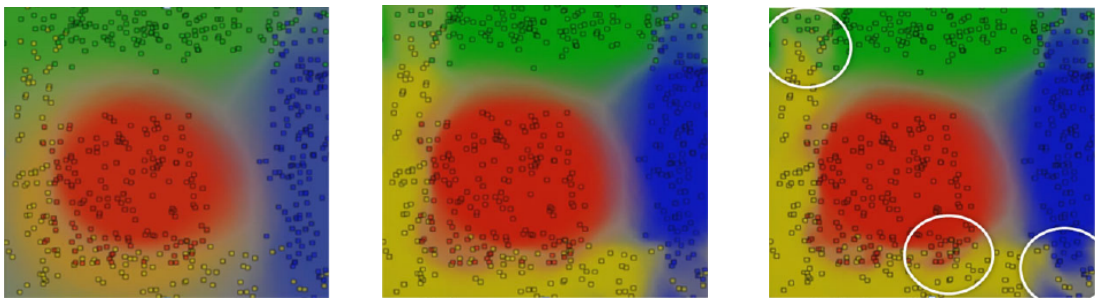


(a) B=1 (b) B=8 (c) B=200

Obr. 1.3: Obrázek zobrazuje pravděpodobnostní funkci klasifikace do dvou tříd. Červená barva znamená pravděpodobnost 1 klasifikace do první třídy, žlutá barva znamená pravděpodobnost 1 klasifikace do druhé třídy. B je počet použitých klasifikačních stromů. (zdroj: [5])

Vliv hloubky stromu

Hloubka stromu je parametr, který udává maximální počet rozvětvení na cestě mezi kořenem a listy stromu. Hloubka stromu ovlivňuje také počet pozorování, které spadají pod jeden list stromu. Pokud je hloubka stromu příliš velká, nastává jev zvaný *přetrénování* (over-fitting), kdy strom sice dokonale pokryje všechna pozorování, ovšem není schopen správně predikovat výstupy pro jiné než trénovací vstupy. Znakem přetrénování je zahrnování šumu trénovacích dat do modelu stromu (viz. Obr. 1.4(c)). Naopak pokud je hloubka stromu příliš malá, nastává *nedoučení* (under-fitting), kdy strom není dostatečně velký k tomu, aby zachytil důležité struktury trénovacích dat (viz. Obr. 1.4(a)). Výsledkem je to, že strom není schopen správně predikovat ani trénovací ani testovací vstupy.[5]



(a) D=3 (under-fitting) (b) D=6 (c) D=15 (over-fitting)

Obr. 1.4: Obrázek zobrazuje pravděpodobnostní funkci klasifikace do čtyř tříd. D je hloubka stromů. (zdroj: [5])

1.4 Předzpracování obrazů

1.4.1 Proč předzpracovávat?

Předtím, než bude možné použít zadané snímky pro trénování nebo predikci pomocí strojového učení, je nutné tyto snímky předzpracovat. Použitý klasifikační model (Random Forests) je již z principu velmi závislý na nehomogenitách intenzit v trénovacím a testovacím souboru dat a rozdílů mezi nimi. Tyto nehomogenity a rozdíly se však ve snímcích z magnetické rezonance objevují velmi často, a většinou se jim není možné vyhnout. Proto je nutné, při zpracování obrazů počítačem, využít některé metody, které tyto vady na snímcích potlačují.

Dalším problémem při počítačové segmentaci snímků je vliv šumu. Šum je nevyhnutelnou součástí snímků z magnetické rezonance a jeho přítomnost neblaze ovlivňuje přesnost klasifikačního modelu. Pro odstranění šumu se používá množství filtrů, které potlačují vysokofrekvenční složky kmitočtového spektra snímku.

Nehomogenity intenzit uvnitř snímku

Uvnitř snímku vznikají nehomogenity v intenzitách, které jsou dané fyzickým rozpoložením samotného přístroje MR, jeho okolím a snímaným subjektem. Jsou to:

- *Nehomogenity v magnetickém poli statické a transverzální cívky* - lze řešit vytvořením modelu nehomogenit na známé homogení tkáni (například bíla hmota v mozku). Tento postup je velmi komplikovaný, protože je nutné jej provádět pro každý přístroj zvlášť a také jej opakovat při změně v kalibraci.
- *Nehomogenity zpusobené indukováním elektrického proudu ve snímané tkáni*[8] - pro tuto nehomogenitu není možné vytvořit univerzální model, a je tedy nutné použít metodu, jež bude vytvářet model pro každý získaný snímek zvlášť.
- *Nehomogenity způsobené okolními vlivy* - tyto nehomogenity vznikají indukcí rušivých elektromagnetických signálů na snímacích cívkách a lze je eliminovat dodržováním správných zásad při práci s magnetickou rezonancí.

Rozdíly intenzity mezi snímky

Rozdíly intenzit mezi snímky vznikají kvůli rozdílům v kalibraci přístrojů, poloze subjektu, a kvůli mnoha dalším vlivům, které nelze předvídat. Je proto nutné, před samotnou klasifikací, intenzity ve snímku normalizovat tak, aby odpovídaly snímkům, na nichž byl klasifikační model trénován. Normalizací docílíme toho, že intenzita pixelů určité tkáně v jednom referenčním snímku bude odpovídat intenzitě pixelů stejné tkáně na jiném snímku.[8]

1.4.2 Metoda N3 - odstranění nehomogenit magnetického pole

Nehomogenity v magnetickém poli jsou v této metodě odhadovány s využitím zjednodušujícího modelu, jež rozkládá intenzity snímku na tři části. První částí je samotná odezva tkáně $u(x)$, kterou se snažíme zjistit. Druhou částí je nehomogenita pole $f(x)$, což je pomalu se měnící hladká funkce. Poslední, třetí, částí je šum $n(x)$. Model pak udává, že výsledná intenzita na snímku $v(x)$ je dána tímto vztahem:

$$v(x) = u(x)f(x) + n(x).$$

Dalším zjednodušením, které lze zavést, je zanedbání vlivu šumu, což model zjednodušší na:

$$v(x) = u(x)f(x).$$

Zlogaritmováním tohoto modelu pak dostaneme:

$$\hat{v}(x) = \hat{u}(x) + \hat{f}(x),$$

kde $\hat{v}(x) = log[v(x)]$,$\hat{u}(x) = log[u(x)]$ a $\hat{f}(x) = log[f(x)]$.

Uvažujme nyní o $\hat{v}(x)$, $\hat{u}(x)$ a $\hat{f}(x)$ jako o spojitých náhodných veličinách s hustotou pravděpodobnosti $V(t)$, $U(t)$ a $F(t)$. Pro hustoty pravděpodobností pak platí:

$$V(t) = U(t) * F(t).$$

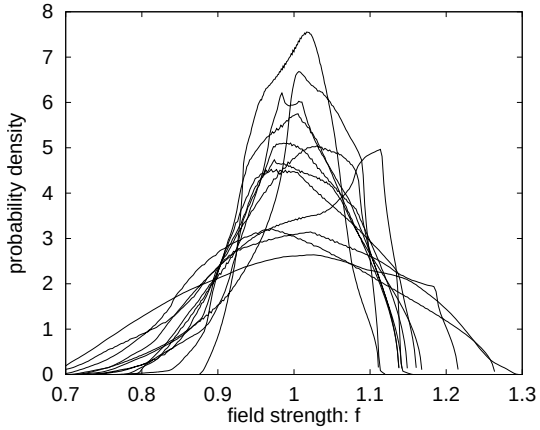
Z grafu na obrázku 1.5 lze vypozorovat, že tvar hustoty pravděpodobnosti $\hat{f}(x)$ připomíná tvar hustoty pravděpodobnosti normálního rozdělení. Z toho vyplývá, že nehomogenita magnetického pole působí na hustotu pravděpodobnosti odezvy tkáně jako dolní propust a potlačuje tak vysokofrekvenční složky.

Využítím faktu, že $F(t)$ lze aproximovat tvarem Gaussovy funkce, můžeme získat odhad $\hat{u}(x)$ pomocí dekonvoluce $V(t)$ právě s Gaussovou funkcí. Tento přístup má však dva problémy:

1. neznáme směrodatnou odchylku σ, jež je parametrem Gaussovy funkce
2. tvar $F(t)$ neodpovídá Gaussově funkci úplně

Řešením těchto problémů je využít iterační přístup, kdy z $V(t)$ dekonvolvujeme úzkou Gaussovou funkci. Tím získáme data pro výpočet odhadu $\hat{f}(x)$, jež je hladkou a pomalu se měnící funkcí. Aplikací tohoto odhadu na $V(t)$ získáme odhad $U(t)$. Dekonvolucí úzké Gaussovy funkce postupně odstaňujeme vliv $F(t)$[2] a zároveň se držíme toho, že výsledné pole bude hladké a pomalu se měnící. Opakováním tohoto postupu se dostaneme do bodu, kdy po sobě následující odhady $\hat{f}(x)$ budou jen minimálně odlišné - v tomto stavu je již nehomogenita odstraněna a iteraci lze ukončit. (zdroj: [8])

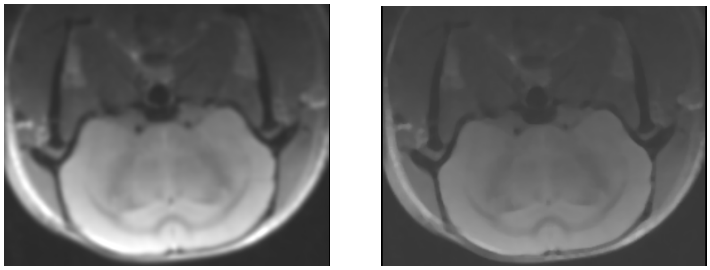
[2] Jelikož jakoukoliv Gaussovu funkci je možné rozložit na konvoluci užších Gaussových funkcí.



Obr. 1.5: Graf zobrazuje hustotu pravděpodobnosti funkce nehomogenity magnetického pole, získané z bílé hmoty na snímku dvanácti subjektů na dvanácti přístrojích MR. (zdroj: [8])

1.4.3 Mediánový filtr - potlačení šumu

V principu je tento filtr velmi jednoduchý, Při iteraci, přes všechny pixely v obrazu, filtr vybere pro každý pixel několik okolních pixelů, seřadí je podle velikosti a prostřední pixel zapíše na aktuální pozici iterátoru. Složitost takovéto implementace by byla $O(r^2 logr)$, kde r je poloměr oblasti pixelů, z nichž se medián vybírá. V praxi je tedy nutné využívat mnohem sofistikovanější metody pracující s poli histogramů. (zdroj: [1])



(a) Gauss (b) Medián

Obr. 1.6: Porovnání výsledku filtrování Gaussovým a Mediánovým filtrem

1.4.4 Histogram matching - normalizace intenzit mezi snímky

Uvažujme o intenzitách na snímku jako o spojité náhodné veličině $v(x)$. Na vstupu algoritmu máme referenční snímek $v_{ref}(x)$ s distribuční funkcí $V_{ref}(t)$ a snímek k normalizaci $v_1(x)$ s distribuční funkcí $V_1(t)$. Cílem *histogram matching* algoritmu je najít

takovou funkci $H(t)$, pro kterou platí:

$$V_{ref}(t) = H(V_1(t))$$

a tuto funkci pak aplikovat na intenzity pixelů ve snímku $v_1(x)$. Výsledkem tohoto algoritmu jsou snímky, jejichž histogram se svým tvarem blíží k tvaru histogramu referenčního snímku. (zdroj: [11])

1.5 Extrakce příznaků

1.5.1 Úvod

Pro klasifikaci vzorku ze segmentovaného snímku používá klasifikační model informaci zvanou *příznak*. Příznak je informace získaná (extrahovaná) určitou metodou ze snímku. Většina příznaků použitých v této práci je možné extrahovat přímo ze zpracováváného snímku. Jeden z těchto příznaků, „hustota hran“ (edge density [10]), však vyžaduje předzpracování snímku algoritmem pro detekci hran. V této práci jsem pro detekci hran použil Sobelův operátor, jehož popis následuje v kapitole 1.5.2.

1.5.2 Sobelův operátor - detekce hran

Sobelův operátor je určený k detekci hran ve snímku. Aplikací tohoto operátoru na snímek dostáváme mapu hran, kde větší intenzita pixelu odpovídá větší síle hrany. Ve tvaru konvolučních jader udává Sobelův operátor jádra (viz. Obr. 1.7, Obr. 1.8), jejichž aplikací na snímek získáme detekci hran v určitém směru. Provedením detekce ve směru os X a Y, získáme pro každý bod na snímku hodnotu G_x a G_y. Sloučením těchto dvou hodnot pak dostaneme sílu hrany G. (zdroj: [11])

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Obr. 1.7: Sobelův operátor ve směru osy X

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ 1 & 0 & 1 \end{pmatrix}$$

Obr. 1.8: Sobelův operátor ve směru osy Y

2 PRAKTICKÁ ČÁST

2.1 Popis zpracovávaných dat

Zpracovávanými daty jsou v této práci MR snímky hlavy myši. Obrazová data jsou uložena ve čtyřrozměrné matici $X \times Y \times Z \times T$, kde T značí časový interval od počátku přechodového děje, po kterém je snímek $X \times Y \times Z$ získán. Z značí číslo řezu a $X \times Y$ značí jednotlivé pixely obrazových dat vybraného řezu. Rozměry zadaných snímků jsou $256 \times 256 \times 21 \times 5$.

Společně s obrazovými daty jsou k zadaným snímkům dostupné také ruční segmentace těchto snímků (tzv. *labely*). Labely jsou uloženy jako původní snímky, ve kterých je bílou barvou vykreslen okraj segmentovaného mozku. Label snímky jsou pouze tří rozměrné a jejich velikost je $256 \times 256 \times 21$.

U všech snímků včetně labelů je použit formát *Analyze 7.5*[12], který vyžaduje pro uložení jednoho snímku dva soubory:

- *Hlavičkový soubor* s příponou „.hdr“
- *Datový soubor* s příponou „.img“

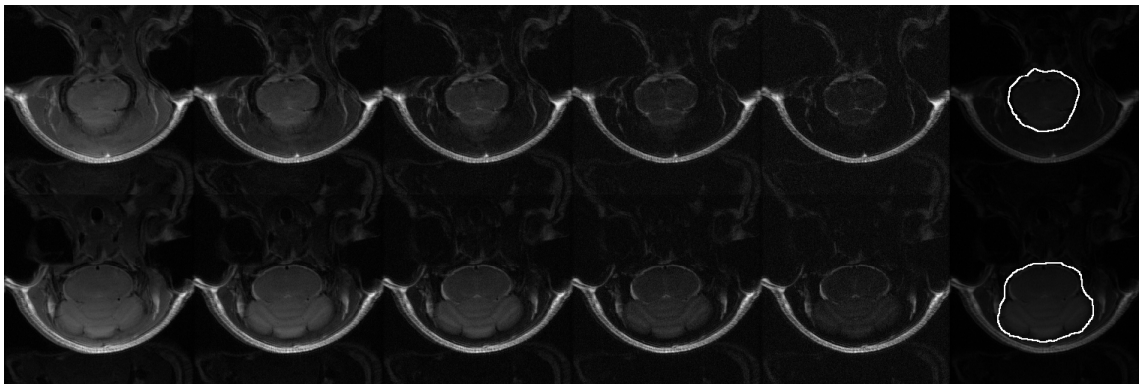
V **hlavičkovém souboru** jsou uloženy všechny informace o snímku. Mezi těmito informacemi je mimo jiné: *počet dimenzí*, *rozměry snímku*, *reálná vzdálenost mezi vzorky* v milimetrech a *reálné umístění počátku snímku* v prostoru. V **datovém souboru** jsou uložena obrazová data. Intenzity obrazu mohou být v tomto souboru uloženy v jednom z následujících formátů:

- *binární* (1 bit na pixel)
- *unsigned char* (8 bitů na pixel)
- *signed short* (16 bitů na pixel)
- *signed int* (32 bitů na pixel),
- *float* (32 bitů na pixel, pohyblivá řádová čárka)
- *komplexní* (64 bitů na pixel)
- *double* (64 bitů na pixel, pohyblivá řádová čárka)
- *RGB* (128 bitů na pixel)

2.2 Použité technologie

2.2.1 ITK (Insight segmentation and registration ToolKit)

ITK je opensource softwarový projekt, který zpřístupňuje třídy a struktury pro práci s medicínskými obrazy. Jde o velmi rozsáhlý projekt umožnující zpracování, registraci a segmentaci medicínských obrazů. *Segmentace* je proces, který identifikuje a klasifikuje data nalezená v digitálně vzorkovaných reprezentacích snímků. *Registrace*



Obr. 2.1: Dva řezy z MR snímku

Poslední snímek zobrazuje ruční segmentaci mozku.

je proces, který upravuje data obrazu tak, aby se podobal jinému obrazu. V medicínském prostředí to znamená např. složení CT a MR obrazu pro získání většího množství informací o zobrazeném objektu.[1]

Jedná se o multiplatformní projekt, který je ze zdrojových kódu sestaven pomocí CMAKE[1], který řeší problémy týkající se jedné specifické platformy, na které je projekt sestavován. Projekt ITK je vyvíjen v jazyce C++ s využitím generického programování. To je způsob, který využívá tzv. templaty k tomu, aby mohl vytvořit zdrojový kód, který je možné genericky při kompilaci aplikovat na jakýkoliv datový typ. To umožnuje využívat třídy ITK na obrazy s jakýkoliv počtem rozměrů. Pracovat se třídami ITK lze užitím velkého množství různých programovacích jazyků (např. C++, Java, Python, ...). Jelikož se jedná o opensource projekt, jakýkoliv vývojář může přispět svým kódem, a tím se ITK neustále vyvíji neuvěřitelnou rychlostí.[1]

Dokumentace[2] projektu je generovaná ze zdrojových kódů pomocí doxygenu [2]. K projektu je dále dostupný manuál [1].

Instalace

Vzhledem k tomu, že pro linuxovou distribuci Slackware, kterou používám, neexistuje balíček s instalací ITK, vytvořil jsem tzv. SlackBuild[3] skript, který automaticky zkompiluje zdrojové kódy a vygeneruje z nich balíček (viz příloha A.1). Před použitím je vhodné tento soubor upravit tak, aby výsledný balíček zahrnoval potřebné moduly.

[1] `http://www.cmake.org/`
[2] `www.doxygen.org`
[3] `slackbuild.org`

Třídy užité v projektu

Pro splnění zadání bylo využito množství tříd, které ITK poskytuje. Zde je jejich soupis a vysvětlení použití.

- itk::Image - kontejner obrazových dat, díky templatování je možné do této jedné třídy vkládat obrazy s různým počtem dimenzí i různou barevnou hloubkou pixelů.
- itk::ImageFileReader - třída umožnující jednoduchým způsobem číst obrázky z různých typů souborů.
- itk::ImageFileWriter - třída umožnující jednoduchým způsobem zapisovat obrázky různých typů do souborů různých typů.
- itk::RescaleIntensityImageFilter - třída umožnující přizpůsobit intenzity jednotlivých pixelů do rozsahů, které lze explicitně definovat.
- itk::ResampleImageFilter - třída umožnující změnit velikost obrazů na specifikovanou velikost.
- itk::SliceBySliceImageFilter - třída umožnující zpracovávat obrazy o n rozměrech po *n-1* rozměrných částech.
- itk::BinaryThresholdImageFilter - třída, která obraz ve stupních šedi převede na binární obraz tím, že intenzity pixelů v zadaném intervalu nahradí zadanou intenzitou a intenzity mimo rozsah nahradí jinou intenzitou.
- itk::BinaryFillholeImageFilter - je třída vyplňující ohraničené díry v binárním obrázku.
- itk::MedianImageFilter - nelineární obrazový filtr, který nahradí intenzitu každého pixelu mediánem sestaveným z jeho sousedících pixelů. V projektu je tento filtr použit pro odstranění šumu.
- itk::ExtractImageFilter - třída umožnující rozdělit vícerozměrný obrázek na dvourozměrný.
- itk::JoinSeriesImageFilter - třída umožnující sloučení několika samostatných snímků do jednoho vícerozměrného snímku.
- itk::StatisticsImageFilter - třída umožnující určit statistické veličiny jako je průměrná intenzita či rozptyl intenzity na zadaném obraze.
- itk::LabelStatisticsImageFilter - třída umožnující určit statistické veličiny na zadaném regionu obrazu.
- itk::BinaryImageToLabelMapFilter - třída umožnující převést binární obraz na label mapu.
- itk::LabelMapToLabelImageFilter - třída převádějící label mapu na label snímek.
- itk::ShiftScaleImageFilter - třída umožnující zpracovat každý pixel pomocí vzorce: $x = (x + c_1) * c_2$.

- itk::BinaryShapeKeepNObjectsImageFilter - třída umožnující ponechat v binárním snímku pouze objekty které vyhovují specifikovanému kritériu.
- itk::HistogramMatchingImageFilter - třída provádějící histogram matching na zadaném snímku.
- itk::N4BiasFieldCorrectionImageFilter - třída implementující metodu N3 pro odstranění nehomogenity magnetického pole ze snímků.
- itk::ShrinkImageFilter - třída sloužící ke zmenšení zadaného snímku v určitém poměru podvzorkováním.
- itk::SobelEdgeDectectionImageFilter - třída pro detekci hran ve snímku pomocí Sobelova operátoru.
- itk::BoxMeanImageFilter - třída sloužící pro výpočet průměrné hodnoty z intenzit okolních pixelů.
- itk::BoxSigmaImageFilter - třída sloužící pro výpočet směrodatné odchylky z intenzit okolních pixelů.

2.2.2 OpenCV

OpenCV je opensource knihovna s BSD licencí pokrývající několik stovek algoritmů pro počítačové vidění. Pro knihovnu existují dvě rozhraní. Jedno je určeno pro programování v jazyku C a druhé, objektové rozhraní, je určeno pro programování v jazyce C++.[7]

OpenCV je rozděleno na několik modulů, z čehož vyplývá, že se nejedná o jednu knihovnu, ale o celou soustavu knihoven, z nichž každá má svůj účel. Jsou dostupné následující moduly:[7]

- *core* - definuje základní datové struktury a základní funkce nad těmito strukturami
- *imgproc* - zpacování obrazů, filtrování, geometrické transformace, histogramy, atd...
- *video* - zpracování videa
- *calib3d* - práce s vícerozměrnými obrazy
- *features2d* - funkce pro detekci hran
- *objdetect* - detekce objektů
- *highgui* - jednoduché uživatelské rozhraní
- *gpu* - algoritmy akcelerované použitím grafického procesoru

Random Forests implementace

Knihovna OpenCV obsahuje vlastní implementaci algoritmu Random Forests, která byla v této práci využita. Tato implementace pracuje se strukturou *CvRTParams*,

která obsahuje parametry, jimiž se řídí trénovací algoritmus.[7] Následuje výčet jednotlivých parametrů:

- *max_depth* - maximální hloubka konstruovaných stromů
- *min_sample_count* - minimální počet pozorování zahrnutých pod jedním listem stromu
- *max_categories* - Při konstruování stromů se třídy dělí na podskupiny, což napomáhá k větší efektivitě algoritmu. Velikost těchto podskupin bude menší nebo rovno max_categories. Tento parametr se využívá jen u problémů, kde je počet možných tříd větší než dvě.
- *calc_var_importance* - pokud je tento parametr nastaven na hodnotu *true* algoritmus bude při učení počítat důležitost jednotlivých vstupních proměnných
- *nactive_variables* - velikost náhodně vybírané podskupiny ze skupiny vstupních proměnných
- *max_num_of_trees_in_forest* - maximální počet vytvářených stromů
- *forest_accuracy* - maximální odchylka mezi predikcemi jednotlivých stromů
- *termcrit_type* - specifikuje pomínku, za které se učení stromu ukončí

2.3 Implementace vlastního projektu

Pro implementaci projektu jsem zvolil jazyk C++, vzhledem k předchozím zkušenostem s tímto jazykem. Dále bylo nutné zvolit knihovnu, která umožnuje práci s medicínskými obrazy. V této oblasti nebylo nutné příliš vybírat, toolkit ITK je asi jedinou volbou. Výběr vhodné implementace *Random Forests* již není tak jednoznačný. Mezi možnostmi byly knihovny Sherwood[4] a OpenCV[5]. Zvolena byla knihovna OpenCV, z důvodu lepších licenčních podmínek. Samotný problém projektu lze rozdělit na 6. části:

1. Předzpracování MR obrazů.
2. Předzpracování ručních segmentací.
3. Trénování klasifikačního modelu.
4. Predikce pomocí natrénovaného modelu.
5. Zpracování výstupů predikce.
6. Porovnání získané segmentace s ruční segmentací.

[4] `http://research.microsoft.com/en-us/projects/decisionforests/`
[5] `http://opencv.org/`

2.3.1 Předzpracování obrazů

Načtení snímků

Prvním krokem při zpracování obrazů je načtení těchto dat. Při tomto kroku jsem využil ITK třídu ImageFileReader. Nejprve jsem definoval typ se specifikací typu pro template třídy ImageFileReader. Dále jsem vytvořil instanci této třídy a nastavil jméno souboru, ze kterého se bude číst. Intenzity pixelů jsou v objektu obrázku uloženy jako datový typ float, což umožnuje provádět na snímku výpočty s pohyblivou řádovou čárkou.

```
typedef itk::ImageFileReader<VolImageType> ReaderType;
ReaderType::Pointer reader = ReaderType::New();
reader->SetFileName(argv[1]);
```

Odstranění nehomogenit magnetického pole

Druhým krokem při předzpracování obrazu je odstranění nehomogenit magnetického pole ze snímku. K tomuto účelu jsem využil vylepšnou implementaci *N3* metody, která je implementována v ITK ve třídě *itk::N4BiasFieldCorrection*[9]. Tato metoda nepotřebuje na vstupu segmentovaný snímek, narozdíl od většiny ostatních metod, a tím je umožněno její využití v jednom z prvních kroků předzpracování snímků.

Vzhledem ke složitosti této metody je vhodné snímek nejprve převzorkovat na menší formát. V této práci jsem snímky převzorkoval na $\frac{1}{4}$ původní velikosti ve směru os X a Y. Dalším zvoleným zjednodušením je to, že výpočet nehomogenity pole provádím pouze na trojrozměrném snímku (T=0). Vzhledem k tomu, že všechny snímky na časové ose byly získány jedinou excitací, je možné odhad pole získaný na jednom trojrozměném snímku aplikovat na ostatní snímky v časové ose. Jelikož výsledný odhad pole je polynomiální funkcí je možné tento odhad namapovat na snímek původní velikosti. Tím tedy bude algoritmus pracovat pouze na snímku o velikosti $64 \times 64 \times 21 \times 1$, avšak ve výsledku neztratíme žádná data a dostaneme snímek o původní velikosti $256 \times 256 \times 21 \times 5$.

Pro zpracování touto metodou je také důležité vytvořit masku, která je použita pro odstranění vlivu pozadí snímku. Přesnost masky není příliš důležitá, ovšem je vhodné nějakou masku použít. V této práci jsem tuto masku vytvořil maskováním pixelů jejichž intenzita je menší než průměrná intenzita snímku, přičemž maska byla vytvořena jen pro první čas měření (T=0).

Na obrázku 2.2 je zobrazen snímek před opravou nehomogenity pole, odhad nehomegenit a také snímek s odstraněným vlivem nehomogenit. V oblasti mozku je jasně vidět, že spodní část původního snímku je světlejší než část horní, což je

potvrzeno zobrazeným odhadem pole. Ve výsledném snímku je již oblast mozku homogenní.

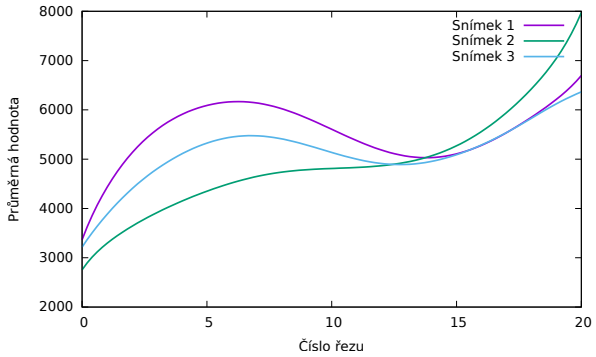
(a) Puvodní snímek (b) Odhad nehomogenity (c) Opravený snímek

Obr. 2.2: Ukázka použití metody N3 na jediný řez

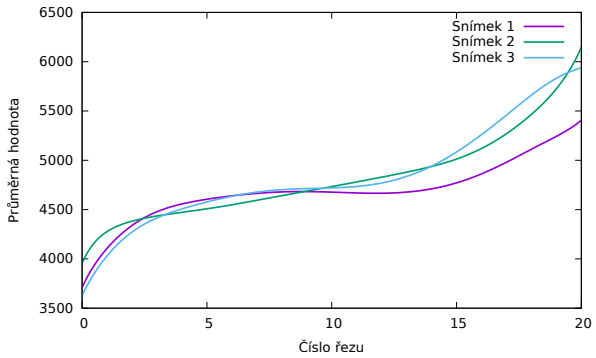
Na grafech 2.3(a) a 2.3(b) jsou zobrazeny průměrné intenzity v jednotlivých řezech ručně segmentované oblasti mozku, před a po aplikaci N3. Na grafech 2.4(a) a 2.4(b) je zobrazen rozptyl intenzit v jednotlivých řezech ručně segmentované oblasti mozku, před a po aplikaci N3. Z těchto grafů lze usoudit, že přestože metoda zjevně účině odstranila nehomogenity v jednotlivých řezech, na nehomogenity mezi řezy působí omezeně.

Nejdůležitějšími parametry této metody jsou:

- *Number of fitting levels* - tento parametr udává složitost výsledné funkce odhadu. Je nutné jej volit experimentálně. Pokud je v odhadu vykreslena struktura zpracovávaného snímku, je třeba tuto hodnotu snížit. V této práci jsem zvolil hodnotu *3*.
- *Convergence threshold* - udává podmínku pro ukončení iterace. Pokud bude rozdíl mezi dvěmi po sobě jdoucími odhady menší než tato hodnota, je iterace ukončena. V této práci jsem zvolil hodnotu *0.0001*.
- *Maximum number of iterations* - maximální počet iterací. V této práci jsem zvolil hodnotu *50*.



(a) Před aplikací N3

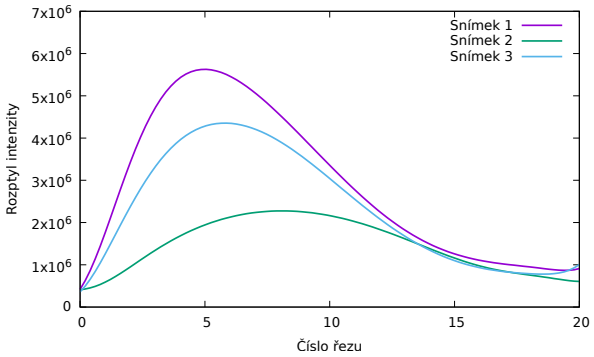


(b) Po aplikaci N3

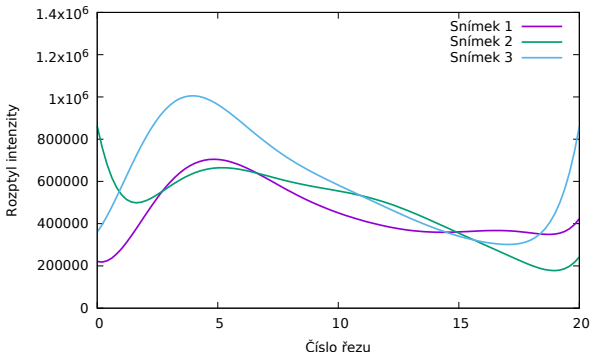
Obr. 2.3: Průměrná hodnota intenzit na oblasti mozku v jednotlivých řezech před (2.3(a)) a po (2.3(b)) aplikaci metody N3.

Dále uvádím ukázku kódu jež provádí opravu zadaného snímku pomocí získaného odhadu pole.

```
itk::ImageRegionIterator<N4Type::ScalarImageType> inIt(
  bspliner->GetOutput(),
  bspliner->GetOutput()->GetLargestPossibleRegion() );

itk::ImageRegionIterator<Image3dType> outIt(
  extractedVolume,
  extractedVolume->GetLargestPossibleRegion() );

float exp;
for(
  inIt.GoToBegin(), outIt.GoToBegin();
  !inIt.IsAtEnd();
  ++inIt, ++outIt
){
  exp = std::exp( static_cast<float>( inIt.Get()[0] ) );
  outIt.Set(outIt.Get() / exp);
}
```



(a) Před aplikací N3



(b) Po aplikaci N3

Obr. 2.4: Rozptyl intenzit na oblasti mozku v jednotlivých řezech před (2.4(a)) a po (2.4(b)) aplikaci metody N3.

Potlačení vlivu šumu

Dalším krokem je aplikace filtru pro potlačení šumu. K tomuto účelu jsem využil *median filter* implementovaný v ITK. Následuje ukázka kódu, který vytvoří a nastaví instanci filtru. Mediánový filtr jsem v této práci zvolil po testování dopadu mnoha různých filtrů na výsledky klasifikace. Filtr výborně potlačuje šum a jeho výhodou je, že často ponechává hrany objektů nerozostřené.

Jediným parametrem je poloměr jádra, který udává, z jak velkého okolí pixelu bude vybrán medián. Velkým poloměrem riskujeme ztrátu dat, malým poloměrem zase nízkou účinost při potlačení šumu. V této práci jsem zvolil poloměr **2**.

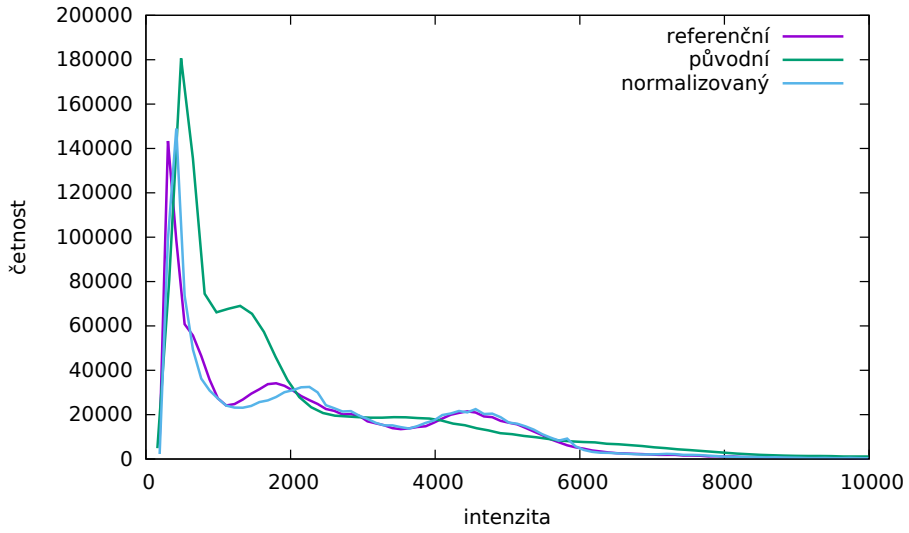
```
typedef itk::MedianImageFilter
  < Slice3dType::InternalInputImageType,
    Slice3dType::InternalOutputImageType >
  MedianType;

MedianType::Pointer medianFilter = MedianType::New();
```

```
medianFilter->SetRadius( radius );
slice3dFilter->SetFilter( medianFilter );
```

Normalizace obrazu

Normalizace obrazu je proces, kdy se snažíme dosáhnout co nejmenšího rozdílu v intenzitě mezi jednotlivými testovacími vzorky. Tento krok je důležitý, protože klasifikační stromy jsou na rozdíly v intenzitě velice citlivé. Pro normalizaci intenzit jsem v této práci použil metodu zvanou *histogram matching*. U této metody je nutné zvolit dva parametry:

- *Number of histogram levels* - udává počet hladin intenzity, které budou použity pro vytvoření histogramu. (zvolena hodnota *100*)
- *Number of match points* - udává v kolika bodech bude histogram snímku upraven, tak aby odpovídal referenčnímu histogramu. (zvolena hodnota *10*)



Obr. 2.5: Histogram referenčního (fialová) a zpracovávaného snímku před (zelená) a po (modrá) aplikaci metody histogram matching.

```
typedef itk::HistogramMatchingImageFilter
  < Slice4dType::InternalInputImageType,
    Slice4dType::InternalOutputImageType > MatchingType;

MatchingType::Pointer matchingFilter = MatchingType::New();
matchingFilter->SetNumberOfHistogramLevels( 100 );
matchingFilter->SetNumberOfMatchPoints( 15 );
matchingFilter->ThresholdAtMeanIntensityOn();
```

2.3.2 Předzpracování labelů

Načtení labelů

Načítání labelů probíhá stejně jako načítání snímků. Jediný rozdíl je, že labely intenzity pixelů jsou načteny do celočíselného typu *signed short*. Jelikož pro trénování potřebujeme, aby každému pixelu ve snímku přiřazoval odpovídající pixel z labelu určitou třídu, je nutné upravit formát labelu. V původním formátu obsahuje label uzavřenou křivku, která vymezuje oblast mozku. Pro další použití je tedy nutné prostor ohraničený touto křivkou vyplnit hodnotou, jež odpovídá třídě náležící oblasti mozku. Samotnou ohraničující křivku je poté třeba z labelu vyloučit.

Po načtení labelu je nutné převést jej na binární obraz. Nejprve bylo třeba zjistit intenzitu pixelů křivky I_L, která označuje mozek. K tomu jsem využil třídu *itk::MinimumMaximumImageCalculator*, která určí minimální a maximální hodnotu intenzity ve snímku. Maximální zjištěná intenzita pak odpovídá intenzitě I_L. Nyní již můžeme aplikovat *itk::BinaryThresholdImageFilter*, který pixely s intenzitou odpovídající I_L nahradí hodnotou **1** a ostatní pixely hodnotou **0**.

```
typedef itk::BinaryThresholdImageFilter
  < in_image_type,
    extract_features::mask_type > ThresholdType;

ThresholdType::Pointer thresholdFilter =
  ThresholdType::New();
thresholdFilter->SetInput( imageReader->GetOutput() );
thresholdFilter->SetUpperThreshold(
  minMaxFilter->GetMaximum() - 1
);
thresholdFilter->SetInsideValue( 0 );
thresholdFilter->SetOutsideValue( 1 );
```

V dalším kroku jsem pak upravoval data pro klasifikační stromy tak, aby všechny pixely mozku byly tímto labelem označeny. To jsem provedl algoritmem pro vyplňování ohraničených děr.

```
typedef itk::BinaryFillholeImageFilter
  < SliceFilterType::InternalInputImageType > FillHoleType;

FillHoleType::Pointer fillHoleFilter = FillHoleType::New();
fillHoleFilter->SetForegroundValue( 1 );
```

Posledním krokem je vyloučení pixelů samotné křivky. Pro tento účel jsem využil *itk::BinaryErodeImageFilter*, který z hranice označené oblasti odstraní stanovený počet pixelů.

```
typedef itk::BinaryErodeImageFilter
  < SliceFilterType::InternalInputImageType,
    SliceFilterType::InternalOutputImageType,
    BallType> ErodeType;

ErodeType::Pointer erodeFilter = ErodeType::New();
erodeFilter->SetInput( fillHoleFilter->GetOutput() );
erodeFilter->SetKernel( ball );
erodeFilter->SetForegroundValue( 1 );
erodeFilter->SetBackgroundValue( 0 );
```

2.3.3 Extrakce příznaků

Použité příznaky

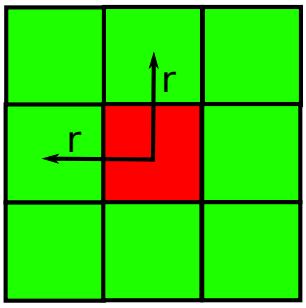
v této práci jsem použil celkem 55 příznaků. použité příznaky lze rozdělit do těchto čtyř skupin:

1. intenzita pixelů
2. intenzita v okolí pixelu
3. textura v okolí pixelu
4. kontext pixelu

v následujících kapitolách bude vysvětlen princip extrakce těchto příznaků. v popisu metod pro extrakci snímku budu používat termín *pixel* jako pixel, ke kterému se vztahují popisované příznaky.

Intenzita pixelů

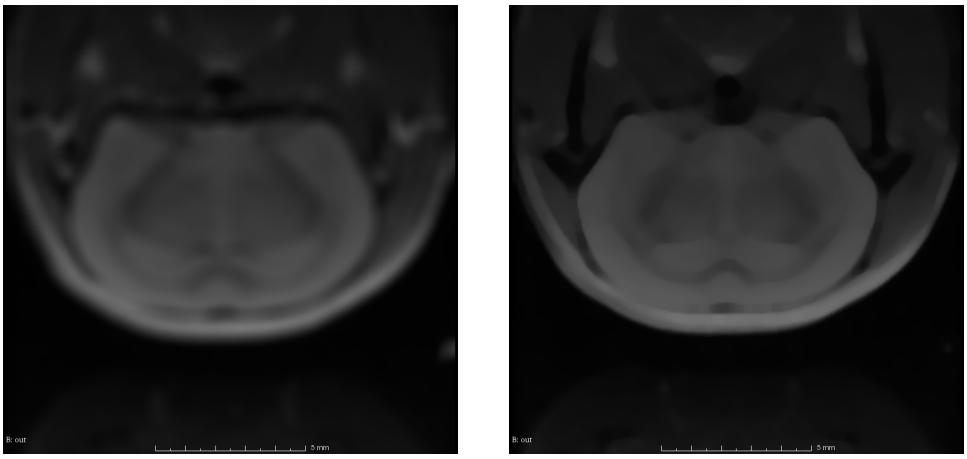
Intenzita pixelů je příznak, který vychází přímo ze zadaných snímků. Pro trojrozměrný snímek by tento příznak nebyl nic jiného než jediná hodnota, která nese o daném *pixelu* pouze informaci o jeho vlastní intenzitě. Vzhledem k tomu, že v této práci jsou vstupní snímky čtyřrozměrné (každý řez byl snímán v pěti časech), získáváme pro každý pixel pět hodnot intenzity. Pro extrakci tohoto příznaku jsem v projektu implementoval metodu *extract_features::intensityFeature()*. K tomuto příznaku není třeba uvádět ukázku, protože by odpovídala zadaným snímkům.



Obr. 2.6: Čtvercové okolí pixelu zadané poloměrem **r**.

Intenzita v okolí pixelu

Pod tuto skupinu spadají v projektu dva příznaky, a to „průměrná hodnota“ a „medián“. Oba tyto příznaky vycházejí ze čtvercového okolí *pixelu* jehož velikost je zadána poloměrem **r**. Příznak „průměrná hodnota“ z pixelů v tomto okolí počítá průměrnou hodnotu intenzity. Příznak „medián“ hledá v seřazeném poli intenzit z daného okolí medián. K extrakci *průměrné hodnoty* slouží v práci metoda *extract_features::meanFeature()* a pro extrakci *mediánu* slouží metoda *extract_features::medianFeature()*. Pro poloměr **r** jsem zvolil u obou příznaků hodnotu **4**.



(a) „průměrná hodnota“ (b) „medián“

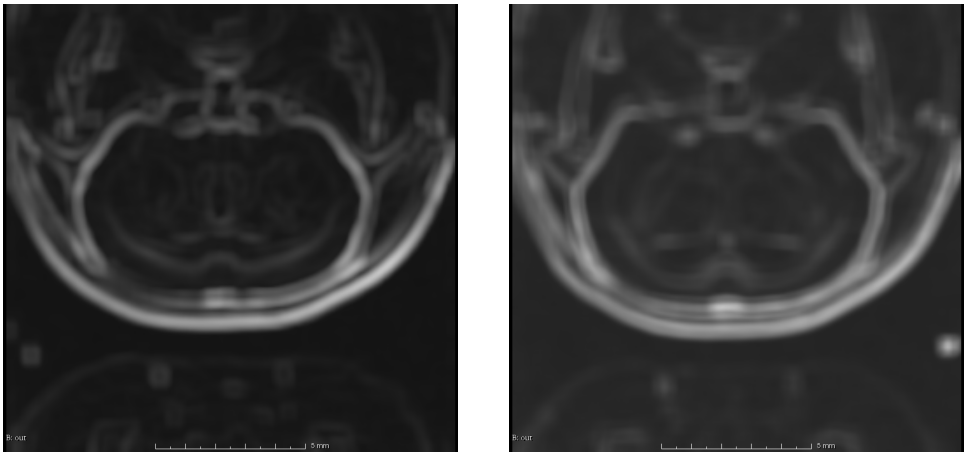
Obr. 2.7: Ukázka příznaků „průměrná hodnota“ a „medián“

Textura v okolí pixelu

Z této skupiny příznaků jsem použil dva, a to *směrodatnou odchylku* a *hustotu hran* v okolí *pixelu*.

Směrodatná odchylka je vypočtena pro hodnoty intenzit ve čtvercovém okolí pixelu jež je dáno poloměrem **r**. K extrakci *směrodatné odchylky* slouží v práci metoda *extract_features::sigmaFeature()*, která využívá třídu *itk::BoxSigmaImageFilter*. Pro tento příznak jsem použil hodnotu poloměru **4**.

Pro určení *hustoty hran* je třeba nejprve použít filtr pro detekci hran. V této práci jsem použil pro detekci hran *Sobelův operátor* (viz. kapitola 1.5.2). Pro extrakci *hustoty hran* slouží v práci metoda *extract_features::edgeDensityFeature()*, která využívá třídu *itk::SobelEdgeDetectionImageFilter*. Pro tento příznak jsem použil hodnotu poloměru **4**.



(a) *směrodatná odchylka* (b) *hustota hran*

Obr. 2.8: Ukázka příznaků *směrodatná odchylka* a *hustota hran*

Kontext pixelu

Z této skupiny jsem použil příznak, který vypovídá o průměrné hodnotě intenzity v určité vzdálenosti od daného *pixelu*. Tento příznak je jako jediný získáván z trojrozměrného okolí, čímž zpřístupňuje informaci o blízkých řezech. Tento příznak zahrnuje celkem 6 hodnot pro každý trojrozměrný snímek na ose T (celkem tedy 30 hodnot). Všechny tyto hodnoty jsou vypočteny jako průměrná hodnota intenzity v šesti kvádrech složených z pixelů snímku. Tyto kvádry jsou vzhledem k *pixelu* umístěny následujícím způsobem:

- Vlevo, v rovině XY, vzdálený o d_1 bodů.
- Vpravo, v rovině XY, vzdálený o d_1 bodů.
- Nahoře, v rovině XY, vzdálený o d_1 bodů.
- Dole, v rovině XY, vzdálený o d_1 bodů.
- Nahoře, v rovině XZ, vzdálený o d_2 bodů.
- Dole, v rovině XZ, vzdálený o d_2 bodů.

K extrakci kontextových příznaků v práci slouží metoda *extract_features::contextFeature()*. Parametry d_1 a d_2 jsem zvolil takto: $d_1 = 9$, $d_2 = 2$.

2.3.4 Převod datových struktur z ITK do OpenCV

Před tím, než je možné extrahované příznaky předat trénovacímu algoritmu, je nutné tyto příznaky upravit do správné formy. Implementace Random Forests v OpenCV požaduje jako trénovací data dvě matice. V jedné matici jsou jednotlivé příznaky ke každému pozorování a v druhé matici je klasifikace daného příznaku. Je tedy nutné implementovat metodu, která převede příznaky z *itk::Image* [6] na matici[7]. Tomuto účelu slouží metoda *extract_features::intensityFeature()*, která zároveň slouží i k extrakci prvního příznaku. Ostatní metody pro extrakci příznaků pak pracují tak, že původní snímek upraví takovým způsobem, aby intenzity pixelů v tomto snímku odpovídaly hodnotě daného příznaku. Takto upravený snímek předají metodě *extract_features::intensityFeature()*, z níž získají výslednou matici příznaků. Výsledné matice, získané extrakcí všech příznaků, jsou následně sloučeny a předány buďto trénovací, nebo predikční metodě.

```
cv::Mat extract_features::intensityFeature
(
  image_type::Pointer image,
  mask_type::Pointer mask
)
{
// Get input image region and size
  image_type::RegionType imgRegion =
    image->GetRequestedRegion();
  image_type::SizeType imgSize = imgRegion.GetSize();
  image_type::IndexType imgIndex;

// Get input mask region
  mask_type::RegionType maskRegion =
    mask->GetRequestedRegion();
  mask_type::IndexType maskIndex;

// Allocate output matrix
  unsigned long n_samples =
    imgSize[0] * imgSize[1] * imgSize[2];
```

[6] *itk::Image* je třída ITK reprezentující N rozměrný obraz.

[7] Maticí je myšlena instance třídy *cv::Mat*, která reprezentuje dvourozměrnou matici.

```
  unsigned long n_features = imgSize[3];
  cv::Mat out( n_samples, n_features, cv_type );

// Create mask iterator
  typedef itk::ImageRegionConstIteratorWithIndex
    < mask_type > MIteratorType;
  MIteratorType mit( mask, maskRegion );

// Iterate through image and mask at the same time
  int row = 0, col; // Index into the matrix

  for( mit.GoToBegin(); !mit.IsAtEnd(); ++mit )
  {

#ifndef IGNORE_MASK
    if( mit.Get() != 0 ){ // Exclude masked pixels
#endif

      // Set the pixel intensities to mat
      maskIndex = mit.GetIndex();
      imgIndex[0] = maskIndex[0];
      imgIndex[1] = maskIndex[1];
      imgIndex[2] = maskIndex[2];

      for( col = 0; col < n_features; ++col ){
        imgIndex[3] = col;
        out.at< cvType >( row, col ) =
          image->GetPixel( imgIndex );
      }

      ++row; // feature sets linearly in rows

#ifndef IGNORE_MASK
    } // if
#endif

  } // for

  // Extract only portion that contains
```

```
  // feature sets ( second arg is exclusive )
  return out.rowRange( 0, row );
} // intensityFeature()
```

2.3.5 Trénování

Poté, co jsou data ve správné formě, je třeba vytvořit strukturu parametrů pro trénovací funkci. Nejdůležitějšími parametry jsou maximální hloubka stromu (max depth) a maximální počet použitých stromů (max number of trees in the forest). Zjištění vhodných hodnot pro tyto parametry je nutné provést pomocí testování na zadaných datech. Pro trénování modelu jsem použil následující parametry:

- Maximální hloubka stromů: **20**.
- Minimální počet vzorků v konečném listu stromu: **5**.
- Počet příznaků náhodně vybraných z vektoru všech příznaků: **40**.
- Počet stromů v modelu: **30**.
- Priorita tříd: 1, 1. (obě třídy mají stejnou prioritu)
- Kritérium ukončující trénování: počet stromů.

```
CvRTParams params = CvRTParams(
  20, // max depth
  5, // min sample count
  0, // regression accuracy: N/A here
  false, // compute surrogate split, no missing data
  2, // max number of categories
  priors, // the array of priors
  false,  // calculate variable importance
  0,
  30,     // max number of trees in the forest
  0.0001f, // forrest accuracy
  CV_TERMCRIT_ITER
  // termination cirteria
);
```

Po vytvoření této struktury je možné zahájit trénování na datech.

```
rtree->train(train_attrs, CV_ROW_SAMPLE, train_clsf_int,
                Mat(), Mat(), var_type, Mat(), params);
```

2.3.6 Predikce

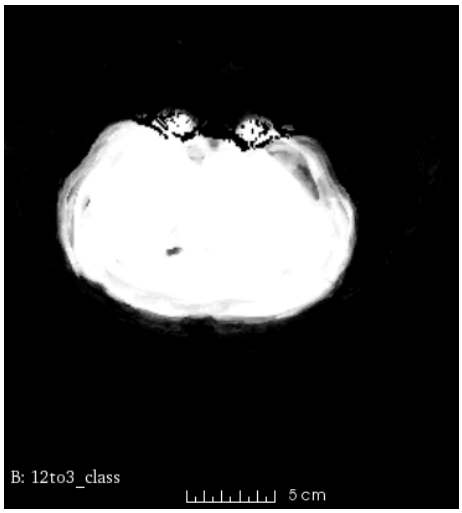
Po dokončení trénování modelu je již možné provést predikci pro testovací data. Predikci je nutné provést iterací přes všechny body v zadaném obraze. Výsledkem bude buďto přímá klasifikace daného bodu, nebo pravděpodobnost s jakou bod patří do druhé třídy. Při přímé predikci klasifikace použijeme metodu *CvRTrees::predict()*, která nejdříve získá predikci všech stromů v modelu a výslednou třídu stanoví jako třídu, kterou navrhuje většina stromů. Při predikci pravěpodobnosti použijeme metodu *CvRTrees::predict_prob()*, která vrátí hodnotu odpovídající počtu stromů, jež navrhují druhou třídu v poměru k celkovému počtu stromů. V této práci jsem zvolil druhou metodu, která umožňuje další zpracování výsledků.

Pro účel predikce jsem v programu napsal funkci *predictor()*, která pro vstupní vektor příznaků vrací pravděpodobnost klasifikace do druhé třídy. Ukazatel na tuto funkci pak předávám metodě *extract_features::reconstructLabel()*, která vygeneruje výstupní snímek.

```
float predictor( void *arg, cv::Mat features )
{
  CvRTrees *rtree = static_cast< CvRTrees * >( arg );

  return rtree->predict_prob( features );
}
```

2.3.7 Zpracování predikcí

Pokud bychom chtěli z výstupní predikce pravěpodobnosti získat klasifikaci, nejjednodušší by bylo přiřadit hodnotám větším než 0.5 druhou třídu (mozek) a ostatním hodnotám první třídu (pozadí). Tímto bychom získali výsledek stejný jako při predikci klasifiace metodou *CvTrees:predict()*. Když však máme snímek s predikcí pravděpodobnosti, je možné na tento snímek aplikovat Gaussův filtr, a tím zvýšit pravěpodobnost klasifikace do druhé třídy u pixelů, které se nacházejí blízko oblasti mozku. Výsledná segmentace tak bude mít hladší hrany a bude spojitější (viz obr.2.9).



Obr. 2.9: Pravděpodobnost klasifikace do druhé třídy

Dalším krokem zpracování bude převedení pravděpodobností na binární snímek (label). V této práci jsem k tomu použil třídu *itk::BinaryThresholdImageFilter*.

```
typedef itk::BinaryThresholdImageFilter
  < image_type, mask_type > ThresholdType;
ThresholdType::Pointer thresholdFilter =
  ThresholdType::New();
thresholdFilter->SetInput( gaussFilter->GetOutput() );
// Classifier model returns values between 0 and 1
thresholdFilter->SetUpperThreshold( 0.5 );
thresholdFilter->SetOutsideValue( 1 );
thresholdFilter->SetInsideValue( 0 );
```

Následně jsem provedl odstranění malých oblastí mimo mozek, které byly klasifikovány jako druhá třída. K tomu jsem využil třídu *itk::BinaryShapeKeepNObjectsImageFilter*.

```
typedef itk::BinaryShapeKeepNObjectsImageFilter< mask_type >
  KeepNType;
KeepNType::Pointer keepNFilter = KeepNType::New();
keepNFilter->SetInput( thresholdFilter->GetOutput() );
keepNFilter->SetAttribute(
  KeepNType::LabelObjectType::NUMBER_OF_PIXELS
);
keepNFilter->SetFullyConnected( true );
keepNFilter->SetNumberOfObjects( 1 );
keepNFilter->SetForegroundValue( 1 );
```

```
keepNFilter->SetBackgroundValue( 0 );
```

Posledním krokem zpracování je vyplnění děr v labelu. K tomu jsem využil třídu *itk::BinaryFillholeImageFilter*.

```
typedef itk::BinaryFillholeImageFilter< mask_type >
  FillHoleType;

FillHoleType::Pointer fillHoleFilter = FillHoleType::New();
fillHoleFilter->SetInput( keepNFilter->GetOutput() );
fillHoleFilter->SetForegroundValue( 1 );
```

2.4 Výsledky segmentace

Testování projektu jsem prováděl celkem na třech snímcích metodou „leave-one-out“, využitím jednoho snímku pro testování a zbývajících dvou pro trénování. Výsledný label jsem poroval s ruční segmentací, a to výpočtem *celkové úspěšnosti*, množstvím *falešne negativních* a množstvím *falešně pozitivních* klasifikací pixelů.

Celkovou úspěšnost **E** jsem vyhodnotil jako počet správně klasifikovaných pixelů v poměru s celkovým počtem pixelů ve snímku tedy:

$$E = \frac{1}{N}(N - \sum_{i=1}^{N} I[(P_i - L_i) = 0]), \tag{2.1}$$

kde P_i je i-tý pixel predikovaného labelu, L_i je i-tý pixel labelu ruční segmentace a I je predikát, který v případě splnění podmínky reprezentuje hodnotu 1 a v případě nesplnění 0. N je celkový počet pixelů na snímku. Výsledkem je kladná hodnota menší nebo rovna 1 (E), kde 1 značí stoprocentní úspěšnost a 0 stoprocentní chybovost.

Množství *falešně negativních* klasifikací jsem vyhodnotil jako počet pixelů klasifikovaných jako třída „pozadí“, které jsou v labelu ruční segmentace klasifikovány jako „mozek“, v poměru s celkovým počtem pixelů.

Množství *falešně pozitivních* klasifikací jsem vyhodnotil jako počet pixelů klasifikovaných jako třída „mozek“, které jsou v labelu ruční segmentace klasifikovány jako „pozadí“, v poměru s celkovým počtem pixelů.

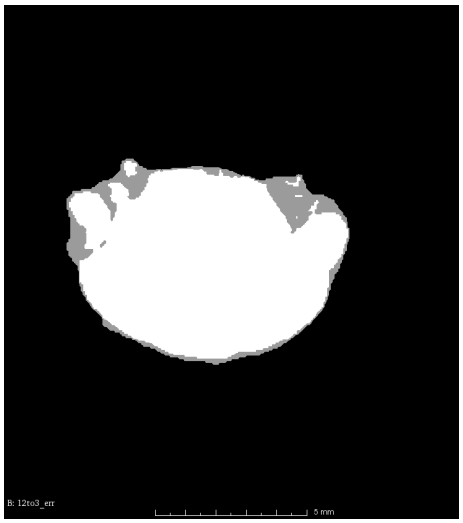
Parametry pro trénování			
max. hloubka	**min. počet vzorků v konečném listu**	**počet náhodně vybraných příznaků**	**počet stromů**
20	5	40	30

Parametry pro extrakci příznaků			
průměrná hodnota - r	**medián - r**	**směrodatná odchylka - r**	**hustota hran - r**
4	4	4	4
kontext pixelu - d1	**kontext pixelu - d2**	**kontext pixelu - rozměry kvádru**	-
9	2	11 x 11 x 3	-

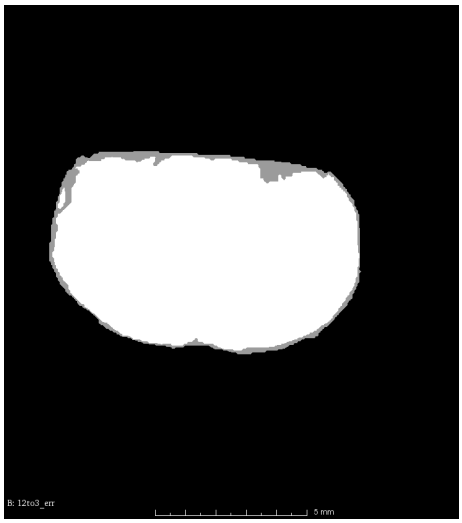
Tab. 2.1: Použitá kombinace parametrů pro trénování a testování

S použitím parametrů uvedených v tabulce 2.1 jsem dosáhl následujících výsledků:

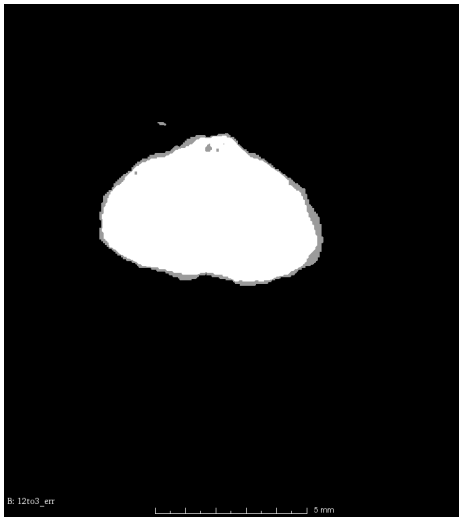
- Trénovací snímky: 1,2. Testovací snímky: 3.
  Celková úspěšnost: 98.2797%
  Falešně pozitivní: 0.2567%
  Falešně negativní: 1.4635%
- Trénovací snímky: 1,3. Testovací snímky: 2.
  Celková úspěšnost: 98.6727%
  Falešně pozitivní: 0.4313%
  Falešně negativní: 0.8960%
- Trénovací snímky: 2,3. Testovací snímky: 1.
  Celková úspěšnost: 98.6667%
  Falešně pozitivní: 0.2026%
  Falešně negativní: 1.13075%



(a) Predikce - řez 1



(b) Predikce - řez 2



(c) Predikce - řez 3

Obr. 2.10: Snímky zobrazují porovnání labelu získaného predikcí s ruční segmentací. Šedá barva označuje místa kde se tyto dva labely neshodují.

3 ZÁVĚR

V této práci jsem se věnoval segmentaci mozku myši ze snímků MR pomocí ML algoritmu *Random Forests*. Pro trénování a testování modelu *Random forests* jsem použil tři snímky myši a jejich ruční segmentace mozku. Použité parametry pro získávání příznaků a pro trénování modelu jsou vypsány v tabulce 2.1. Výsledky testování jsou vypsány v kapitole 2.4 a na obrázku 2.10. V následujících odstavcích jsou shrnuty poznámky ohledně dosažených výsledků.

U krajních řezů snímků dochází k velkému výskytu falešně negativních pixelů. To je zřejmně způsobeno rozdílem intenzit mezi prostředními a krajními řezy. Tato situace by se zřejmně dala řešit použitím některé metody pro normalizaci intenzit mezi řezy. U řezů 15 a 16 dochází k velkému výskytu falešně pozitivních pixelů, většina z nich je však odstraněna pozdějším zpracováním. Velkým problémem pro trénování modelu je také nepřesnost ručních segmentací. Nepřesnosti ručních segmentací se podepisují na jistotě klasifikačního modelu a také na důvěryhodnosti výsledků testování. Přesnost natrénovaného modelu je také značně omezena šumem ve snímcích, který s vyšším časem T narůstá. Řešením tohoto problému by bylo použití většího poloměru protišumového filtru s větším časem T.

Dalšího zlepšení výsledků by se dalo dosáhnout přidáním snímků získaných T1 sekvencí do trénovacích dat. Vzhledem k omezenému souboru testovacích snímků, nelze tvrdit, že jsou stanovené parametry optimální. Pro reálné použití by bylo třeba trénovat model na větším množství snímků.

LITERATURA

[1] *ANALYZE 7.5 File Format* [online]. [cit. 2015-05-28]. <http://eeg.sourceforge.net/ANALYZE75.pdf>.

[2] CRIMINSI, Antonio - SHOTTON, Jamie. *Decision forests for computer vision and medical image analysis*. 1st ed. New York: Springer, 2013. ISBN 978-144-7149-286.

[3] FISHER, Robert - PERKINS, Simon - WALKER, Ashley - WOLFRAT, Erik. *Hypermedia Image Processing: Morphology* [online]. [cit. 2015-5-28]. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm>.

[4] GONZALEZ, Rafael - WOODS, Richard. *Digital Image Processing*. Second edition. New Jersey, USA: Prentice-Hall, Inc., 2002. ISBN 0-201-18075-8.

[5] HANSON - Lars. *Introduction to Magnetic Resonance Imaging Techniques* [online]. 2009 [cit. 2014-12-14]. <http://eprints.drcmr.dk/37/1/MRI_English_a4.pdf>.

[6] HASTIE, Trevor - TIBSHIRANI, Robert - FRIEDMAN, Jerome. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction: Data Mining, Inference, and Prediction*. 2nd ed. Stanford, California: Springer, 2008. ISBN 978-0387848570.

[7] INSIGHT SOFTWARE CONSORTIUM. *Insight Toolkit Documentation* [online]. 2014 [cit.2014-12-11] <http://www.itk.org/Doxygen46/html/index.html>.

[8] JOHNSON, Hans - MCCORMICK, Matthew - IBANEZ, Luis. *The ITK Software Guide Book 1: Introduction and Development Guidelines* [online]. Fourth Edition. 2014 [cit. 2014-12-11] <http://itk.org/ItkSoftwareGuide.pdf>.

[9] OPENCV DEVELOPMENT TEAM. *OpenCV documentation* [online]. 2011 [cit. 2014-12-14]. <http://docs.opencv.org/index.html>.

[10] PHUNG, Son Lam - BOUZERDOUM, Abdesselam. *Detecting People in Images: An Edge Density Approach*. In *IEEE, International Conference on Acoustics, Speech and Signal Processing, 2007*. Honolulu, Hawaii, USA: Conference Management Services, Inc. 2007. 1, I-1229-I-1232.

[11] SLED, John - Alex ZIJDENBOS - Alan EVANS. IEEE. *A Nonparametric Method for Automatic Correction of Intensity Nonuniformity in MRI Data*. In

IEEE TRANSACTIONS ON MEDICAL IMAGING [online]. 1998, 17(1): 87-97 [cit. 2015-05-21]. <https://www.nitrc.org/docman/view.php/6/880/sled.pdf>.

[12] TUSTISON, Nicholas - James GEE. PICSL, UNIVERSITY OF PENNSYLVANIA. *N4ITK: Nick's N3 ITK Implementation For MRI Bias Field Correction.* In *The Insight journal* [online]. 2010 [cit. 2015-05-21]. ISSN 2327-770x. <http://hdl.handle.net/10380/305>

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

MR	Magnetická Rezonance
ML	strojové učení – Machine Learning
CT	počítačová tomografie – Computed Tomography
ITK	Insight segmentation and registration ToolKit

SEZNAM PŘÍLOH

A OBSAH ELEKTRONICKÉ PŘÍLOHY

A.1 Popis souborů

- Soubor „InsightToolkit.SlackBuild“ je shellový skript sloužící pro kompilaci ITK ze zdrojových kódů.
- Složka „src“ obsahuje zdrojové kódy projektu. Pro sestavení projektu je třeba postupovat podle kapitoly A.2.
- Soubor „model.xml“ je záznam popisující strukturu predikčního modelu.

Po sestavení projektu budou ve složce „src/build“ dostupné tyto spustitelné soubory:

- Program „compare_labels“ slouží k porovnání predikovaného labelu s předzpracovanou ruční segmentací. Jeho výstupem je snímek zobrazující rozdíl mezi těmito dvěmi labely.
- Program „postprocess“ slouží ke zpracování predikce z programu „train“.
- Program „predict“ slouží k segmentaci snímku pomocí klasifikačního modelu.
- Program „preprocess“ slouží k předzpracování snímku pro predikci nebo trénování.
- Program „preprocess_label“ slouží k predzpracování labelů pro trénování nebo porovnávání.
- Program „train“ slouží k trénování klasifikačního modelu.

Spuštění kteréhokoliv z uvedených programů s parametrem „-h“ vypíše nápovědu k programu.

A.2 Sestavení projektu

Pro sestavení projektu je třeba mít funkční systém GNU/Linux a na něm nainstalovány knihovny ITK a OpenCV. Pro sestavení je také nutné mít nainstalován CMAKE.

Pokud jsou uvedené závislosti vyřešené, můžeme program sestavit. Nejprve ve složce „src“ vytvoříme novou složku „build“ a vstoupíme do ní.

```
$ mkdir src/build
$ cd src/build
```

Dalším krokem je konfigurace projektu.

```
$ cmake ..
```

Nyní již můžeme projekt sestavit.

```
$ make
```

Pokud vše proběhlo v pořádku měli bychom mít v aktuální složce spustitelné soubory „compare_labels“, „postprocess“, „predict“, „preprocess“, „preprocess_label“ a „train“.